# pwdgen

Srini, rs@toprllc.com

August 22, 2021

## 1 Introduction

With the proliferation of services available on the web and the corresponding need to use distinct passwords, we all face challenges coming up with unique passwords. This projectlet helps generate passwords that meeting typical requirements e.g. letters and numbers and special characters and so on.

A related challenge is to remember the passwords so generated and a companion projectlet builds a password manager built around this generator.

This projectlet is the starting point for a series of projectlets culminating in a pair of secure servers to generate and store the passwords.

## 2 Implementation

### 2.1 Libraries

Some features of this projectlet such as key derivation utilize the **OpenSSL** libraries (`https://wiki.openssl.org/index.php/Main_Page`).

### 2.2 Approach

The example implementation uses a list of words and chooses one at random. The projectlet supplies a default list of words but a user can feed a list in a text file. Instead of a meaningless sequence of letters, a list of words may be easier to remember. An engineer may prefer a list of words related to engineering and the sciences while a musician might prefer to feed a list of words drawn from a musical dictionary.

Similarly the numerical components are also random numbers.

A **segment** is composed of one word and a number concatenated using a supplied separator - typically a special character.

# 3 Examples

## 3.1 Usage

```
$ obj/pwdgen --help
pwdgen V01 2021-08-22 16:52:58
Usage: pwdgen.exe [switches] [password to derive key for]

 -v, --verbose                 Output extra verbose information
 -s, --segments=ARG            Number of Segments
 -m, --max-word-length=ARG     Maximum length of words
 -f, --word-list-file-name=ARG Word list file name
 -p, --separator=ARG           Separator
 -i, --iterations=ARG          Iterations for key derivation
 -d, --dump-spec               Dump ada spec
 -b, --builtin-wordlist        use builtin wordlist
 -k, --derive-key              derive key
```

## 3.2 Examples using the built-in word list

**One segment** Each invocation generates a different random password consisting of a built in word and a number.

```
$ obj/pwdgen --builtin-wordlist --segments=1
star-29660
$ obj/pwdgen --builtin-wordlist --segments=1
thoracic-55658
$ obj/pwdgen --builtin-wordlist --segments=1
peptic-30298
```

### Multiple segments

```
$ obj/pwdgen --builtin-wordlist --segments=2
perfect-29787-Dendrite-38203
$ obj/pwdgen --builtin-wordlist --segments=3
vein-47519-Bone-52061-jimsonweed-16299
$ obj/pwdgen --builtin-wordlist --segments=3 --separator=~
protein~57038~Nettle~33965~virus~20685
$ obj/pwdgen --builtin-wordlist --segments=3 --separator=~
azalea~52030~Perfect~58393~thyroid~62850
```

**Key Derivation** Key derivation function generates a fixed length key given a cleartext password. The complexity could be increased by asking for multiple iterations.

```
$ obj/pwdgen --derive-key BasicPassowrd
Derived Key for BasicPassowrd is
bd36e7c15961bdc478f4f4537bc6769c029fc308f8cdad755a17fcd3c62b7f88
$ obj/pwdgen --derive-key BasicPassowrd
Derived Key for BasicPassowrd is
bd36e7c15961bdc478f4f4537bc6769c029fc308f8cdad755a17fcd3c62b7f88
$ obj/pwdgen --derive-key BasicPassowrd --iterations=8
Derived Key for BasicPassowrd is
ab0ec9e1f24768b3e4787d87c8cbad8b7f603569a8df0212c49635acf529d587
```

Implementation in **go** `https://gitlab.com/ada23/pwdgen.git`